1-15-2020

# DeepMotions : A Deep Learning System for Path Prediction Using Similar Motions

Mohammed Abdalla

Abdeltawab Hendawi

Hoda M.O. Mokhtar

Neveen Elgamal

John Krumm

*See next page for additional authors*

## Authors

Mohammed Abdalla, Abdeltawab Hendawi, Hoda M.O. Mokhtar, Neveen Elgamal, John Krumm, and Mohamed Ali

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 16, 2019, accepted December 16, 2019, date of publication January 15, 2020, date of current version February 7, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.2966982*

# *DeepMotions*: A Deep Learning System for Path Prediction Using Similar Motions

**MOHAMMED ABDALLA[1], ABDELTAWAB HENDAWI[2], HODA M. O. MOKHTAR[3], NEVEEN ELGAMAL[3], JOHN KRUMM[4], AND MOHAMED ALI[5]**

[1]Faculty of Computers and Artificial intelligence, Beni-Souef University, Beni Suef 62111, Egypt
[2]Department of Computer Science and Statistics, The University of Rhode Island, Kingston, RI 02881, USA
[3]Faculty of Computers and Artificial intelligence, Cairo University, Giza 12613, Egypt
[4]Microsoft Research, Redmond, WA 98052, USA
[5]School of Engineering and Technology, University of Washington, Seattle, WA 98195, USA

Corresponding author: Mohammed Abdalla (mohammed.a.youssif@fcis.bsu.edu.eg)

**ABSTRACT** Trajectory prediction techniques play a serious role in many location-based services such as mobile advertising, carpooling, taxi services, traffic management, and routing services. These techniques rely on the object's motion history to predict the future path(s). As a consequence, these techniques fail when history is unavailable. The unavailability of history might occur for several reasons such as; history might be inaccessible, a recently registered user with no preceding history, or previously logged data is preserved for confidentiality and privacy. This paper presents a Bi-directional recurrent deep-learning based prediction system, named *DeepMotions*, to predict the future path of a query object without any prior knowledge of the object historical motions. The main idea of *DeepMotions* is to observe the moving objects in the vicinity that have similar motion patterns of the query object. Then use those similar objects to train and predict the query object's future steps. To compute similarity, we propose a similarity function that is based on the KNN algorithm. Extensive experiments conducted on real data sets confirm the efficient performance and the quality of prediction in *DeepMotions* with up to 96% accuracy.

**INDEX TERMS** Deep learning, moving objects, neural network, trajectory prediction.

## I. INTRODUCTION

It is an undeniable fact that GPS-enabled devices such as smart phones and smart watches have become an integral part of nowadays' life. Market studies show that the number of smart phones in usage raised from 1.57 billion in 2014 to be around 2.5 billion in 2018 and expected to reach 2.9 billion in 2020 [45]. A similar but even sharper increasing trend occurs in the smart watches market where the number went from 5 million watches in 2014 to be around 141 million units in 2018 [46]. These GPS-enabled devices have an ubiquitous influence on Location-aware services for the present time, e.g. store locators, as well as for the future time, e.g., traffic prediction.

Future time location-based services, several existing works predict the possible future paths of moving objects to provide a better understanding of human mobility [4], [15], [19], [48]. The majority of this work mainly requires the historical trajectories data of the moving object to compute the anticipated paths. However, the related work in this area has several deficiencies including; (1) fail to predict when the query object's history is insufficient for generating predictions, (2) consider assumptions like objects following the shortest path or frequent trajectory patterns, which usually fails and turns to be non-realistic in several scenarios, (3) perform poorly when the dataset is large, and (4) produce inaccurate predictions in many scenarios.

This paper develops a novel system named *DeepMotions*, to predict the future paths, whereas the query object's historical trajectories data is absent.

The main idea of *DeepMotions* depends on identifying the $K$ objects that presently move similarly to the query object and use those similar objects to build our prediction model using Bi-directional recurrent deep neural network. To identify $K$ similar objects, *DeepMotions* builds its similarity function based on the KNN machine learning algorithm [8], [38], [41]. Therefore, the similarity distance is computed between the query object and other moving objects in the vicinity. It is important to note that the lower the distance value, the greater the similarity between the query object and

The associate editor coordinating the review of this manuscript and approving it for publication was Seok-Bum Ko.

the other object. Here, KNN is fed by two chosen coefficients; *common sub-sequence* and *edit distance*. The *common sub-sequence* score means the number of commonly visited road edges between two trajectories, while the *edit distance* score refers to the number of edits, (i.e., delete, add, or replace of edges), required to make two trajectories exactly like each other. After recognizing the $K$ similar objects that closely follow the query object motion pattern, *DeepMotions* obtains their complete trajectories to form the training dataset of the prediction model. In addition to its role in identifying objects with similar motions, the similarity function deployed in *DeepMotions* plays a significant role in removing irregularities in data and reduces the computational overhead by filtering out trajectories that are less likely to participate in the prediction process. Subsequently, these similar trajectories are fed to a deep learning model which generates potential next path for the query object and their likelihoods.

The major contributions of this paper are the following:

- We introduce a novel deep-learning based model for predicting moving object's future paths named *DeepMotions*. *DeepMotions* is considered as a first approach based on deep neural network notions that predicts the future path of the query object without any prior knowledge about the object's self-history.
- We confirm the generation of highly accurate predictions by adopting the Bi-directional recurrent deep neural network that preserves the information from both past and future motion sequences.
- We devise a novel Edit Distance function to measure the similarity between two trajectories. We provide an empirical evidence that ensures run-time optimization of this function.
- We implement a novel similarity function which omits the irrelevant input features. This function leads to a substantial increase in the *DeepMotions* efficiency by removing irregularities in input data and preventing overfitting issue to occur during the prediction process.
- We propose a KNN algorithm to select input features, this algorithm makes the model works effectively when the data set becomes very large.
- We conduct extensive experiments on real-life datasets. Experimental results prove that our model significantly outperforms the accuracy of competitive methods by 20%.

The rest of this paper is organized as follows. Section 2 studies related work and explores different directions in the area of trajectory prediction. Section 3 formally defines the problem. Section 4 gives a brief review of existing recurrent neural network architectures. The architecture of the *DeepMotions* system is described in Section 5. Section 6 experimentally evaluates *DeepMotions*. Finally, Section 7 concludes the paper.

## II. RELATED WORK

This section reviews the previous research works related to trajectory prediction. The related work is categorized into two categories, namely, historical-based prediction and non historical-based prediction.

### A. HISTORICAL-BASED PREDICTION

In this category, the employed prediction function uses object historical data to predict the object's next location.

#### 1) HMM-BASED PREDICTION

Several models have applied the Hidden Markov model to predict the future locations and routes that a user might go to next. Authors in [20]–[22] implement several algorithms for predicting the route of a vehicle based on the observations of the vehicle's past trips. In [33], the authors introduce an idea for traffic flow prediction in a city area based on Variable-order Markov Model. Authors in [16] propose a model named *R2-D2* which is an HMM-based probabilistic model that predicts the future trajectory path.

#### 2) RNN-BASED PREDICTION

Other attempts tried predicting future paths based on recurrent neural networks notions. notions [7], [9], [11], [27], [28], [31], [51], [53]. In [27], authors propose a novel method named T-CONV which models trajectories as two-dimensional images and adopts multi-layer convolutional neural networks to combine multi-scale trajectory patterns to obtain highly precise predictions and extract the areas with distinct influence on the ultimate prediction. In [9], authors introduce a neural network approach to handle the destination prediction of a taxi based on the beginning of its trajectory and associated metadata. This approach employs a recurrent bidirectional neural network to encode the prefix, several embeddings to encode the metadata and destination clusters to generate the output. In [53], the authors introduce a neural network model named *RA-LSTM* which combines the road-aware features to predict the future trajectory path. Several techniques employing LSTM (Long Short-Term Memory) cells for predicting future trajectories have been introduced in the literature [7], [11], [28], [31], [51].

#### 3) PATTERN-MATCHING PREDICTION

Authors in [54], perform the process of mining frequent trajectory patterns from large-scale trajectories and use those patterns to predict the most probable location of moving objects. In [24], authors propose a method for predicting moving object's future paths. This method retrieves the trajectories whose moving patterns are similar to that of a query trajectory from past trajectories of moving objects stored in databases.

### B. NON HISTORICAL-BASED PREDICTION

#### 1) ASSUMPTION-BASED PREDICTION

Authors in [1], [13] design the Panda system for answering spatial predictive queries on top of moving objects included in Euclidean spaces. In [32], authors support predictive queries based on the assumption that the query object follows the

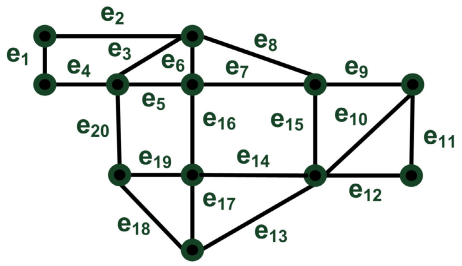



**FIGURE 1.** Road network graph.

shortest route to a destination. Authors in [5], [18], [37], [49], [50] introduce several query processing techniques where the applied prediction function depends on an assumption that objects move in a linear function in time along with the input velocity and direction.

#### 2) INDEX-BASED PREDICTION

In [32], the authors propose an index that supports issuing predictive queries in the absence of the objects' historical trajectories. In [12], authors design an algorithm for trajectory prediction named FFTPA consists of an R-tree index and moving object location updates. Authors in [36] introduce a general framework for predicting and indexing moving objects with unknown motion patterns. Authors in [26] propose an indexing technique that supports efficient querying of moving object future positions.

*DeepMotions* differs from the above studies in the following: (1) *DeepMotions* is considered as the first deep learning model try to perform the prediction process without any knowledge of the query object's past history, (2) *DeepMotions* can scale up when the input datasets become large, (3) DeepMotions guarantees the efficient processing time of queries by filtering out irrelevant input features, and (4) *DeepMotions* produce accurate prediction results compared with other competitive methods.

## III. PROBLEM SETTING

This section demonstrates the preliminary concepts that will be used throughout the paper and highlights the problem statement.

### A. PRELIMINARIES

*Definition 1:* We consider *road network graph* $G = (N, E)$ undirected, where (1) $N$ is a finite set of nodes (node $n \in N$); (2) $E$ is a set of edges which represent the road segments (edge $e \in E$); (3) each node $n$ in $N$ represents the road intersections. Figure1 represents an example of a road network graph.

*Definition 2:* Trajectory $\tau$ is an ordered sequence of edges(road segments) traversed by a moving object over the road network, i.e., $\tau = \{e_1, e_2, e_3, \ldots .e_n\}$.

*Definition 3:* Common sub-sequence $CSS(\tau_i, \tau_j)$ measures the number of same common edges visited by $\tau_i$ and $\tau_j$. The Common sub-sequence match between $(\tau_i, \tau_j)$ is as follows: $|\tau_i \cap \tau_j|$.

**TABLE 1.** Notations.

| symbols | notations |
|---|---|
| $G = (N, E)$ | road network graph |
| $\tau$ | trajectory |
| $CSS(\tau_i, \tau_j)$ | common sub-sequence edges between $\tau_i$ and $\tau_j$ |
| $E_{Dist}(\tau_i, \tau_j)$ | edit distance between between $\tau_i$ and $\tau_j$ |
| $Q_\tau$ | query object's trajectory |
| $\lambda$ | the latest # of edges moved by $Q_\tau$ |
| $\gamma$ | # of future edges that $Q_\tau$ needs to predict |

*Definition 4:* Edit Distance $E_{Dist}(\tau_i, \tau_j)$ measures the number of edits required to make $\tau_i$ exactly similar to $\tau_j$. The formula of edit distance is as follows:

$$E_{Dist}(\tau_i, \tau_i) = \max(length(\tau_i), length(\tau_j)) - CSS(\tau_i, \tau_j) \quad (1)$$

*Definition 5:* Number of Previously Moved Edges $\lambda$ refers to the latest number of edges traversed by query object's trajectory over the road network. For example: assume $\tau_i = \{e_1, e_2, e_3, e_4\}$ and $\lambda = 2$, this means that the Prediction Building Module will be queried by $(e_3, e_4)$.

*Definition 6:* Number of Next Predicted Edges $\gamma$ refers to the number of future edges that query object's needs to predict.

The notations of this paper are summarized in Table 1.

### B. PROBLEM STATEMENT

Let $S$ be a trajectory set of the currently moving objects, $S = \{\tau_1 \ldots .\tau_{|S|}\}$, $\forall \tau_i \in S$, $\tau_i$ is defined as a series of edges, $\tau_i \implies (e_1, \ldots .e_n)$. Let $Q_\tau$ be the query moving object's trajectory which can be expressed as a series of edges moved by the query object $Q_\tau \implies (e_1, \ldots .e_n)$. let $\gamma$ is the number of future edges to predict. Given $Q_\tau, S, \gamma$, the goal is to predict the $\gamma$ future motions of the query object. Assume the $Q_\tau$ historical movements is $\phi$. Our objective is to achieve a high prediction accuracy over very large datasets, in addition, performing efficiently by reducing the query processing time.

## IV. BACKGROUND

The proposed learning model in this work is implemented using artificial Neural Networks (NNs), particularly feedback neural networks. Neural networks [3], [10], [44], [47] are computer programming techniques that excel in performing tasks that are difficult to perform using traditional techniques. Systems that employ neural networks are capable of learning on their own and adapting to changing conditions. In general, there are two types of Neural Networks (NNs); (1) feed-forward networks and (2) feedback networks. Feed-forward NNs allow signals to travel one way only, from input to output, there is no feedback(loops), i.e. the output of any layer doesn't affect that same layer. On the other hand, feedback networks allow signals to travel in both directions by introducing loops in the network, their state changing dynamically until reaching to equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback networks are also referred to as recurrent that denotes to feedback

connections in a single layer. On the whole, the NN layer types are three named respectively; input layer, hidden layer, and output layer. The method used in NNs that calculates the gradients which are needed in the calculation of weights is called backpropagation [17], [23], [40]. Indeed, the backpropagation algorithm is used to search for weight values that minimize the total error of the network over the training datasets. In general, RNNs have three main and common issues, (1) maintaining the states is very expensive., (2) vanishing gradient, and (3) exploding gradient.

*Summary:* To sum up, vanishing gradient and exploding gradient are common issues in RNN. To overcome these issues, Long Short-Term Memory units (LSTM) [14] and Gated Recurrent Units (GRU) [6] are different network architectures that have been designed to solve those issues.

## V. *DeepMotions:* PROPOSED APPROACH

This section illustrates the proposed system (*DeepMotions*).

### A. MAIN IDEA

The main idea behind *DeepMotions* system is to tackle the challenges encountered by the existing prediction techniques as we reviewed in the previous sections, in addition, to predict the future path of the query moving object when its movement's history is hard to obtain. First, the *DeepMotions* system compares the query object's trajectory with other objects which are currently moving in the system and obtains only similar moving objects which move like the query object. Second, based on these similar trajectories, *DeepMotions* extracts the latent motion patterns. After that, *DeepMotions* employes a deep learning model named Prediction Building Module which effectively builds motion dependencies and generates predictions. The output from Prediction Building Module is a hash-table data structure that contains the moved edges by similar trajectories as keys ($\lambda$), and possible motions with probabilities as values ($\gamma$), the length of the key and the value is identified by the end-user of *DeepMotions* system. Finally, *DeepMotions* query the Prediction Building Module by last motions $\lambda$ moved by the query object and infers the future path as the answer to the query.

Figure 2 presents the architecture of *DeepMotions* system which comprised of three main modules namely, Similarity Module, Prediction Building Module, and Path Extractor Module.

### B. ALGORITHM

Algorithm 1 illustrates the pseudo-code of *DeepMotions*. The algorithm receives, (a) query object's trajectory $Q_\tau$, (b) current moving objects trajectories $Trajectories_{Current}$, (c) number of previous moved edges $\lambda$, (d) a number of next predicted edges $\gamma$. As output, the algorithm returns the query object's future path. The algorithm has three major steps that are briefly discussed as follows:

**Step 1: Similarity Check.** The objective of this step is to extract from $Trajectories_{Current}$ only similar trajectories that have similar motion patterns like the query object, and non-similar trajectories are omitted from $Trajectories_{Current}$. In line 6, the algorithm inspects the similarity between the query object's trajectory and the current moving objects' trajectories through a proposed function which will be described later in section 5.1.

**Step 2: Prediction Building Module Creation.** The objective of this step is to create the *DeepMotions* Prediction Building Module based on the similar trajectories generated from step1. This created module is represented by a hash-table called $P_{Table}$ (line 8). The key of this hash-table is a sequence of visited road edges ($e_1, e_2, e_3, ..e_n$), the length of this sequence is determined by the input parameter $\lambda$ and the value of this hash-table is all possible future edges and their corresponding probabilities. Constructing the Prediction Building Module is achieved through a generic function which will be detailed in section 5.2.

**Step 3: Path Extractor Module.** The objective of this step is to return the predicted path along with its relevant probability $P$ as an answer result to the query object. Based on the query object's trajectory and the number of future edges $\gamma$ that the query object needs to move, a look-up operation is executed on the $P_{Table}$ to retrieve a possible predicted path with the highest probability value. The details of this look-up operation are discussed in section 5.3.

**Algorithm 1** *DeepMotions:* Similarity-Based Prediction

1: **INPUT:** Query object's trajectory $Q_\tau$, Current moving objects trajectories $Trajectories_{Current}$, Number of Previous Moved Edges $\lambda$, Number of Next Predicted Edges $\gamma$
2: **SET** $SimilarTrajectoriesList\ S \leftarrow \phi$
3: **SET** $QueryResult\ QR \leftarrow \phi$
4: **SET** $P_{Table} \leftarrow \phi$
5: **/* Step 1: compute similarity */**
6: $S = \text{COMPUTE\_SIMILARITY}(Q_\tau, Trajectories_{Current})$
7: **/* Step 2: generate the Prediction Building Module*/**
8: $P_{Table} = \text{CREATE\_PREDICTION\_BUILDER}(S, \lambda)$
9: **/* Step 3: predict the future path */**
10: $QR = GET\_FUTURE\_PATH(Q_\tau, P_{Table}, \lambda, \gamma)$
11: **OUTPUT:** Return $QR$

### C. SIMILARITY MODULE

#### 1) MAIN IDEA

The main idea of the *Similarity Module* is to extract trajectories of objects that move similar to the query object and neglects non-similar trajectories. In a nutshell, the *Similarity Module* computes the similarity distance between the query object and other moving objects. More specifically, the *Similarity Module* is designed based on KNN algorithm, so the result from the similarity function is $K$ similar objects which move like the query object. To choose $K$ similar objects, the *Similarity Module* computes the common subsequence and edit distance between the query object's trajectory and other trajectories. Hence, the *Similarity Module* computes the edit distance value in terms of the common subsequence

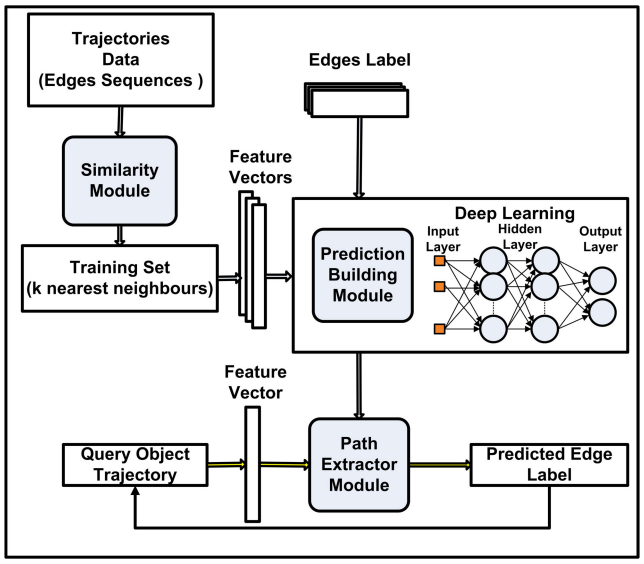



**FIGURE 2.** ***DeepMotions*** **architecture.**

**Algorithm 2** Similarity Module

1: **procedure** Compute_Similarity
2: **INPUT:** Query object's trajectory $Q_\tau$, Moving object's trajectories $Trajectories_{Current}$, $K$ Nearest Objects
3: /* *Nearest neighbour list* */
4: **SET** *NearestNeighbourList* $NNL \leftarrow \phi$
5: **for** *each* $\tau_i$ *in* $Trajectories_{Current}$ **do**
6: /* *Common sequence edges* */
7: $CSS = CSS(Q_\tau, \tau_i)$
8: /* *Excludes non-similar trajectories* */
9: **if** $CSS > 0$ **then**
10: *Compute the* $E_{Dist}(Q_\tau, \tau_i)$ *based on CSS value*
11: *Insert* ($\tau_i$, $E_{Dist}$) *to NNL*
12: **end if**
13: **end for**
14: *Sort NNL by* $E_{Dist}$ *in Ascending order.*
15: **end procedure**
16: **OUTPUT:** Return K objects from *NNL*

value. Then, the *Similarity Module* considers the edit distance value as the final similarity distance. Finally, the *Similarity Module* sorts trajectories in an ascending order based on their associated edit distance values and selects the top $K$ trajectories.

#### 2) ALGORITHM

Algorithm2 presents the pseudo-code of the designed *Similarity Module*. The algorithm takes three input parameters, (a) query object's trajectory $Q_\tau$, (b) current moving objects' trajectories $Trajectories_{Current}$, and (c) $K$, the desired number of nearest objects need to be produced. In line 5, the similarity algorithm iterates over the current moving objects' trajectories. In line 7, the similarity algorithm computes the value of common sub-sequence between query object trajectory and iterated trajectory. In line 9, the value of common sub-sequence is checked to be greater than 0, this condition ensures that non-similar trajectories will be excluded. In line 10, the similarity algorithm computes the edit distance between the query object trajectory and iterated trajectory. The edit distance value is computed according to formula 1. In line 11, the $\tau_i$ and its associated similarity distance are inserted in the nearest neighbor list *NNL*. In line 14, the similarity algorithm sorts the *NNL* in ascending order from the smallest similarity distance to the largest. Finally, the algorithm returns the $K$ nearest objects move like the query object.

#### 3) EXAMPLE

Assume that query object's trajectory is $Q_\tau(e_4, e_5, e_7, e_9)$, other moving objects trajectories are $\tau_1$ to $\tau_8$ with the following sequences $(e_5, e_7, e_9, e_{10})$, $(e_5, e_7, e_9, e_{11})$, $(e_6, e_7, e_9, e_{10})$, $(e_8, e_9, e_{10}, e_{13})$, $(e_7, e_9, e_{10}, e_{14})$, $(e_9, e_{10}, e_{13}, e_{18})$, $(e_9, e_{10}, e_{13}, e_{18}, e_{20})$, $(e_{13}, e_{18}, e_{20})$ and K = 6. First, the Similarity Module iterates over the other moving objects trajectories, for each one the *Similarity Module* computes the common sub-sequence and the edit distance between the query object's trajectory and every iterated trajectory. In the first iteration, the *Similarity Module* computes common sub-sequence between $Q_\tau$ and $\tau_1$ which will be $\text{CSS}(Q_\tau, \tau_1) = 3$ as the longest common sub-sequence between $Q_\tau$ and $\tau_1$ is $(e_5, e_7, e_9)$ which is 3 consecutive edges and computes edit distance between $Q_\tau$ and $\tau_1$ which will be $E_{Dist}(Q_\tau, \tau_i) = 1$ because only one edge edit is needed in order to make $\tau_1$ identical to $Q_\tau$. In the second Iteration, the *Similarity Module* computes the common sub-sequence between $Q_\tau$ and $\tau_2$ which will be $\text{CSS}(Q_\tau, \tau_2) = 3$ as the longest common sub-sequence between the two trajectories is $(e_7, e_9)$ and computes edit distance between $Q_\tau$ and $\tau_2$ which will be $E_{Dist}(Q_\tau, \tau_2) = 1$ because only one edit is needed in order to make $\tau_2$ identical to $Q_\tau$. Similarly, *Similarity Module* computes similarity distances between $Q_\tau$ and other objects. It is important to note that the *Similarity Module* omits $\tau_8$ because it has no common edges with the query object. The corresponding distances between $Q_\tau$ and $\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7$ will be respectively 1, 1, 2, 4, 2, 4, 5. Finally, *Similarity Module* sorts distances ascending and selects the top 6 which will be $\tau_1, \tau_2, \tau_3, \tau_5, \tau_4, \tau_6$.

### D. PREDICTION BUILDING MODULE

#### 1) MAIN IDEA

The main idea behind the proposed Prediction Building Module is creating a data structure that can answer querying objects about their inquiries. As discussed before, Recurrent Neural Networks (RNN) are a family of neural networks that outperforms in learning from sequential data. In this work, the proposed Prediction Building Module is implemented according to RNN architecture, particularly Bi-Directional RNN embedded with LSTM units (BiLSTM) [35], [42], [43], [52]. The justification behind choosing BiLSTM to design Prediction Building Module is that conventional

**Algorithm 3** Prediction Building Module

1: **procedure** CREATE_PREDICTION_BUILDER
2: **INPUT:** Similar Trajectories $SimT$, Number of Previous Moved Edges $\lambda$, Hidden Layers $H_{Layers}$, Layer Nodes $L_{Nodes}$, Road Network $G$, Training Iterations $Trai_{Iters}$
3: /* Dictionary map */
4: **SET** dictionary $dict \leftarrow \phi$
5: /* Reverse dictionary map */
6: **SET** reversedictionary $Rdict \leftarrow \phi$
7: /* The count of unique edges in G */
8: **SET** edgesCount $edges_{Count} \leftarrow \phi$
9: $dict[edge, number] \leftarrow$ map each edge in G to number
10: $Rdict[number, edge] \leftarrow$ Reverse the key/value of dict
11: $edges_{Count} \leftarrow$ select count of distinct edges from G
12: /* Network weights */
13: $Network_{Weights}W \leftarrow Vector[L_{Nodes}, edges_{Count}]$
14: /* Network bias */
15: $Network_{Bias}b \leftarrow Vector[edges_{Count}]$
16: **for** each iter in $Trai_{Iters}$ **do**
17: /* Train the network */
18: $Out1 = forawrdLayer(SimT, W, b, \lambda, H_{Layers})$
19: $Out2 = backwardLayer(SimT, W, b, \lambda, H_{Layers})$
20: **end for**
21: /* Merge the output */
22: $output = BiLSTM(Out1, Out2)$
23: /* Using reverse dictionary to decode the output */
24: $output \leftarrow Mapnumbersinoutputtoedgesby Rdict$
25: **end procedure**
26: **OUTPUT:** Return *output*

LSTM and GRU architectures have a major issue because they are learning sequences from the previous time-stamps only. However, in several cases, it needs to learn sequences from future time-stamps to better understand the context and clears the ambiguity. Figure 3 represents the structure of the BiLSTM, it shows BiLSTM layers and connections; forward layer and backward layer. In the forward layer, connections are going forward in time which learns from previous sequences, in the backward layer connections are going backward in time which learns from future sequences.

### 2) SUMMARY

In Summary, the learning of BiLSTM is done in two steps; (1) moves from left to right starting with the initial time-stamp, computes the values until reaching to the final time stamp, (2) moves from right to left starting with the final time-stamp, computes the values until reaches to the initial timestamp.

### 3) ALGORITHM

Algorithm3 demonstrates the pseudo-code of the *DeepMotions's* Prediction Building Module. The algorithm takes six input parameters, (a) similar Trajectories *SimT* produced by *DeepMotions's* Similarity Module, (b) number of previous moved edges $\lambda$, (c) hidden layers $H_{Layers}$, which represents the number of network layers, (d) layer nodes $L_{Nodes}$, the number of nodes inside each network layer, (e) road network $G$, which is used to extract from it the whole edges, and label them, (f) training iterations $Trai_{Iters}$, which represents how many times the network will be trained to produce the final output. The output from *DeepMotions's* Prediction Building Module is a hash-table that contains the traveled edges of similar trajectories as keys, and expected next edge associated with the prediction probability $P$ as value.

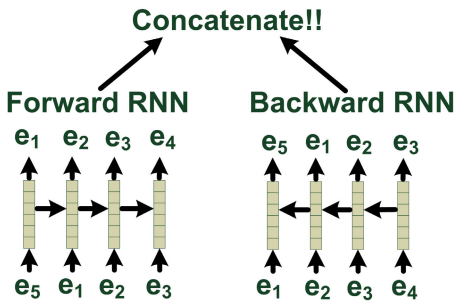



**FIGURE 3.** Bi-Directional RNN.

*DeepMotions's* feeds the BiLSTM by correct sequences $\lambda$ from each trajectory included in similar trajectories list as inputs and one labeled edge, eventually, the network will learn to predict the next edge correctly. Technically, BiLSTM inputs can only understand real numbers. A way to convert the edge to a number is to assign a unique integer to each edge included in the $G$. In line 9, the algorithm builds a dictionary map that contains unique edges in $G$ as key and assigns to them unique numbers as a value. The dictionary map is like the following entries $[e_1 : 0][e_2 : 1], \ldots, [e_3 : 30], \ldots, [e_4 : 311]$. In line 10, the algorithm inverts the key/value pairs of the created dictionary map in line 9 and creates a reversed dictionary map that will be used in decoding the output produced from the BiLSTM. The reversed dictionary map is like the following entries $[0 : e_1][1 : e_2], \ldots, [30 : e_3], \ldots, [311 : e_4]$. In line 11, the algorithm sets the parameter named $edges_{Count}$ with the number of edges in G, this parameter indeed represents the length of the dictionary map. The prediction is a unique integer identifying the index in the reverse dictionary of the predicted edge. For example, if the prediction is 30, the predicted edge is actually $e_3$. The generation of output may sound simple but actually, BiLSTM produces an element vector of size $edges_{Count}$ that contains the probabilities of prediction for the next edge normalized by the softmax() function. The index of the element with the highest probability is the predicted index of the edge in the reverse dictionary (i.e: a one-hot vector). Figure 4 illustrates the process. As shown in Figure 4 each input edge is replaced by its assigned unique integer. The output is a one-hot vector identifying the index of the predicted edge in the reverse dictionary. The weights and biases are defined in

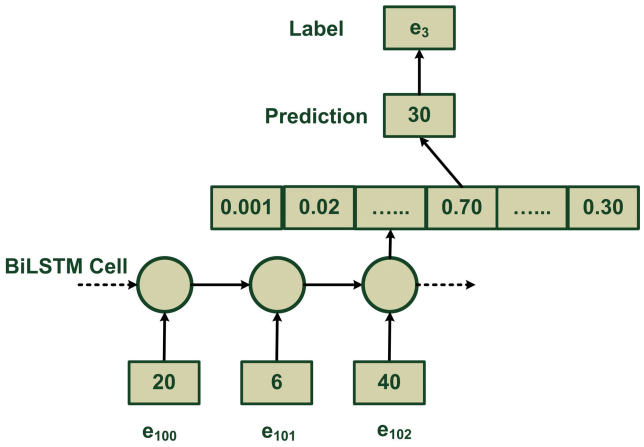



**FIGURE 4.** BiLSTM cell with three inputs and 1 output.

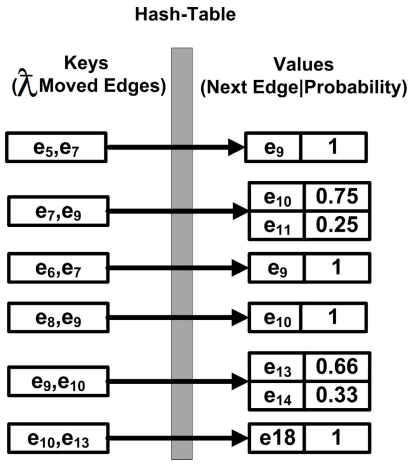



**FIGURE 5.** Prediction building module example.

lines [12-15], they are needed for the network to perform the backpropagation cycle. The algorithm initiates random values for weights and biases. In line 13, the algorithm initiates the weights parameter as matrix of dimensions (number of network nodes' configured per each hidden layer, length of the dictionary map). In line 15, the algorithm initiates the biases parameter as a vector of dimension (length of the dictionary map). In line 16, the algorithm starts the training process. In line 18, the algorithm trains the forward layer and saves the output in variable $Out1$. In line 19, the algorithm trains the backward layer and saves the output in variable $Out2$. In each iteration, the algorithm retrieves $\lambda$ edges from every trajectory included in the $SimT$. The $\lambda$ edges are converted to integers to form the input vector. The training label is a one-hot vector coming from the edge after the $\lambda$ input edges. The accuracy and loss are accumulated to monitor the progress of the training. In line 12, the outputs generated from forward and backward layers are integrated together to produce the final output. Finally, in line 24 algorithm uses the reverse dictionary map to decode the output and returns the final output.

#### 4) EXAMPLE

Figure 5 illustrates an example of the constructed Prediction Building Module as hash-table. Assume the number of previous moved edges $\lambda = 2$, and the similar trajectories list are $\tau_1(e_5, e_7, e_9, e_{10})$, $\tau_2(e_5, e_7, e_9, e_{11})$, $\tau_3(e_6, e_7, e_9, e_{10})$, $\tau_4(e_8, e_9, e_{10}, e_{13})$, $\tau_5(e_7, e_9, e_{10}, e_{14})$, $\tau_6(e_9, e_{10}, e_{13}, e_{18})$

### *E. PATH EXTRACTOR MODULE*

#### 1) MAIN IDEA

The main idea of the Path Extractor Module is to retrieve the predicted path with the highest probability from the Prediction Building Module hash-table. For this purpose, the Path Extractor Module retrieves the last moved edges by the query object, and query the Prediction Building Module using these edges. Consequently, the Prediction Building Module will retrieve the next edge with the highest probability as predicted edge and, adds this predicted edge as part of the expected predicted path. Then, Path Extractor Module feeds back the predicted edge as part of the query object trajectory and loops again to get the next predicted edge. Eventually, when the number of required future edges is equal to the length of the constructed predicted path, the Path Extractor Module breaks the loop and returns the predicted path as a final output to the query object.

#### 2) ALGORITHM

Algorithm4 illustrates the pseudo-code of the proposed Path Extractor Module. The algorithm takes four input parameters, (a) query object's trajectory $Q_\tau$, (b) Prediction Building Module hash-table $P_{Table}$, (c) number of previously moved edges $\lambda$, and (d) number of next predicted edges $\gamma$. In line 5, the Path Extractor Module gets the last moved edges by $Q_\tau$ based on $\lambda$ parameter and sets them as a search key. Next, in line 6 the Path Extractor Module calls $P_{Table}$ by the constructed search key, if the result of this query returns more than one record, the Path Extractor Module gets the record with the highest probability as a result of the query (Lines 6-9). The result of the query contains the next predicted edge and probability. In line 11, the Path Extractor Module adds the predicted edge to the expected predicted path. In line 12, the Path Extractor Module adds the predicted edge to the query object's trajectory and sets it as the new $Q_\tau$. The Path Extractor Module makes a $\gamma$ recursive calls until it predicts the expected $\gamma$ future edges. In line 12, the condition is checked to ensure that the Path Extractor Module will predict the required future edges. Finally, once the length of the constructed predicted path is equal to $\gamma$, the Path Extractor Module returns the predicted path as a final result.

#### 3) EXAMPLE

Assume $Q_\tau(e_4, e_5, e_7, e_9)$, $\lambda = 2$, $\gamma = 3$, and Prediction Building Module hash-table $P_{Table}$ is table constructed in Figure 5. First, Path Extractor Module gets the $\lambda$ moved edges and sets them as a search key $(e_7, e_9)$. Second, the Path Extractor Module manipulates the $P_{Table}$ by $e_7, e_9$. The Path Extractor Module returns the future path with expected

**Algorithm 4** Path Extractor Module

**procedure** Get_Future_Path
**INPUT:** Query object's trajectory $Q_\tau$, Prediction Building Module Hash-Table $P_{Table}$, Number of Previous Moved Edges $\lambda$, Number of Next Predicted Edges $\gamma$
  **SET** *Probability* $\mathcal{P} \leftarrow \phi$
  **SET** *Predicted_Path* $\leftarrow \phi$
  $Search_{Key} \leftarrow$ get last $\lambda$ edges from $Q_\tau$
  $[Predicted_{Edge}, \mathcal{P}] \leftarrow P_{Table}$.get($Search_{Key}$)
  **if** $[Predicted_{Edge}, \mathcal{P}]$.Count( ) $\geq 1$ **then**
    Get$[Predicted_{Edge}, \mathcal{P}]$ with max($\mathcal{P}$)
  **end if**
  Add $Predicted_{Edge}$ To *Predicted_Path*
  $Q_\tau \leftarrow Q_\tau + Predicted_{Edge}$
  **if** *Length of Predicted_Path* $\neq \gamma$ **then**
    Get_Future_Path($Q_\tau$,$P_{Table}$,$\lambda$,$\gamma$)
  **end if**
**end procedure**
**OUTPUT:** Return *Predicted_Path*

probability, so the Path Extractor Module returns the record with the highest probability which is $(e_{10}, 0.75)$. Then, the Path Extractor Module adds the predicted edge to expected future path and to the query object's trajectory and feeds the predicted edge again as part of inputs. As a result, $Q_\tau$ will be $(e_4, e_5, e_7, e_9, e_{10})$. The Path Extractor Module checks if the length of the expected future path is equal to $\gamma$ or not. If the answer is no, the Path Extractor Module will loop again to get the next edge. The Path Extractor Module gets the $\lambda$ moved edges and sets them as a search key $(e_9, e_{10})$. Then, Path Extractor Module query the $P_{Table}$ by $e_9, e_{10}$ and gets $e_{13}$ as the next predicted edge. The Path Extractor Module checks if the length of the expected future path is equal to $\gamma$ or not, the answer is no, so the Path Extractor Module will loop again to get the next edge, adds the predicted edge to the query object's trajectory$Q_\tau(e_4, e_5, e_7, e_9, e_{10}, e_{13})$ and adds the predicted edge to the expected future path $(e_{10}, e_{13})$. The Path Extractor Module gets the $\lambda$ moved edges and sets them as a search key $(e_{10}, e_{13})$. Then, Path Extractor Module query the $P_{Table}$ by $e_{10}, e_{13}$ and gets the $e_{18}$ as the next predicted edge. The Path Extractor Module checks if the length of expected future path is equal to $\gamma$ or not, the answer is yes, so the Path Extractor Module breaks the loop, adds the predicted edge to the expected future path $(e_{10}, e_{13}, e_{18})$ and returns to the query object the constructed future path $(e_{10}, e_{13}, e_{18})$ as the final output.

## VI. EXPERIMENTS

This section evaluates experimentally the accuracy and performance of *DeepMotions* system.

### A. EXPERIMENTAL SETUP

#### 1) DATA-SETS

All the experiments conducted in this work use real-life trajectory data-sets collected by Microsoft Research in the Geo-life project [29] from April 2007 to August 2012. Moreover, these trajectories are divided into small trajectories with an average length of each one 5 road segments. Therefore, the total number of all moving objects trajectories is approximately 2000. The number of previously moved edges $\lambda$ is fixed in our experiments to 4. The average length is configured to be 5 because in the training process, at each step, 4 ($\lambda$) segments are retrieved from the training data, and one segment is labeled as predicted. The road network data is acquired from OpenStreetMap [34]. The data of the road network is obtained from OpenStreetMap [34]. The data of the road network in this work represents Hamilton city in the USA. All trajectories' GPS points (longitude, altitude) are map-matched to road segments(edges) over the road network. Finally, all experiments are conducted on the mapped road segments. The map-matching algorithm is out the scope of this paper [25].

**TABLE 2.** Network parameters.

| Parameter | Value |
|---|---|
| Learning rate | 0.001 |
| Network Depth(# of Hidden Layers) | 2 |
| # of Hidden Nodes in Each Layer | 512 |
| Training Iterations | 10000 |

#### 2) EXPERIMENTAL SETTINGS

The *DeepMotions's* Similarity Module is implemented using Java with JDK 1.8 inside eclipse PHOTONID. The *DeepMotions's* Prediction Building Module is implemented by using python with tensorflow 1.4.0. Tensorflow [39], is a popular open-source python library developed by Google team for implementing deep neural networks or neural networks in general. *DeepMotions's* Prediction Building Module is trained by using network parameters represented in table 2. All experiments are conducted on a PC with Intel(R) Core(TM) i7 processor and 16GB RAM, and running on Windows 10.

#### 3) EVALUATION CRITERIA

This work adopted three metrics to evaluate *DeepMotions* in different phases: (1) loss, (2) accuracy, and (3) CPU processing time. Loss value implies how a specific neural network learning model performs well or poorly after each iteration of optimization. Furthermore, loss value can be considered as the total summation of the errors related to any sample included in the training data set. Commonly, the objective is to reduce the loss value according to the model's network parameters by changing the weight values through backpropagation process in a neural network. The accuracy value is determined as soon as the training samples are fed to the model and log the model results after comparison with true targets. For example, if the number of training samples is 100 and the model deduces 96 of them correctly, so the model accuracy is 96%. Unlike loss, the accuracy value is a percentage, whereas loss is a numeric value. Finally, the third

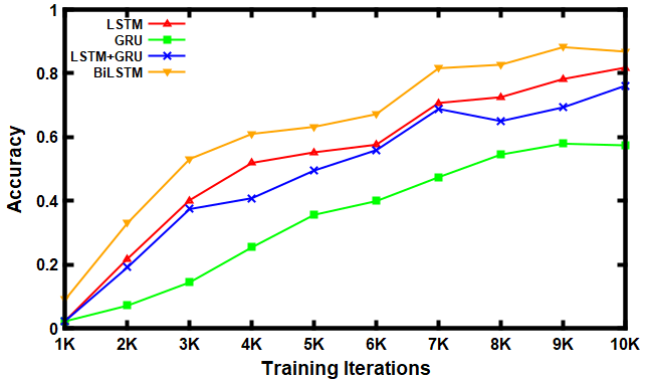
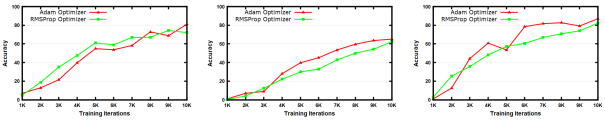
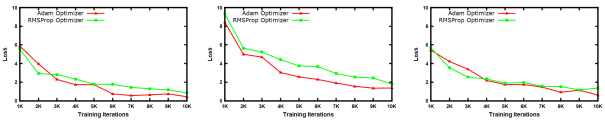



**FIGURE 6.** RNN architectures (accuracy).



(a) LSTM (b) GRU (c) LSTM & GRU

**FIGURE 7.** RNN optimizer (accuracy).



(a) LSTM (b) GRU (c) LSTM & GRU

**FIGURE 8.** RNN optimizer (loss).

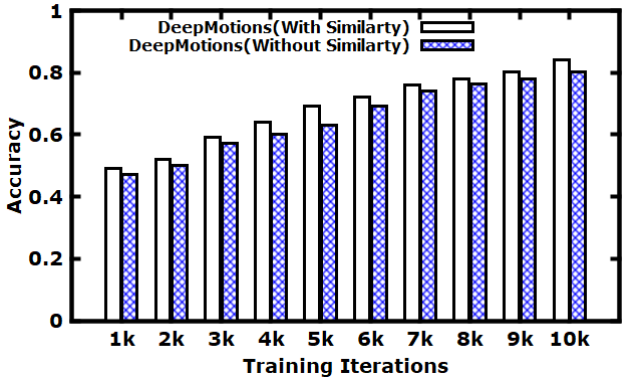



**FIGURE 9.** *DeepMotions* with similarity VS. *DeepMotions* without similarity (accuracy).

metric is the CPU processing time, usually, the main objective is to reach less processing time during the training phase.

### B. ACCURACY AND LOSS EVALUATION

This section illustrates the accuracy and loss evaluations of our proposed system *DeepMotions*. Overall, the accuracy rates show a significant rise over the training iterations, while the loss values experienced a downward trend. As a result, more training iterations means higher accuracy rates, lower loss values, and better model achievement. Also, it is observed that *DeepMotions's* Similarity Module plays a vital role in an accuracy improvement.

### C. ACCURACY AND LOSS EVALUATION RESULTS

We next report our findings.

#### 1) EXP1:- EFFECT OF CHOOSING RNN ARCHITECTURE

In this set of experiments, Figure 6 compares the accuracy of different RNN architectures; BiLSTM, LSTM, GRU, and ensemble architecture named (LSTM + GRU) which make one layer learn using LSTM and another layer learn using GRU. Figure 6 shows the training iterations change from 1000 to 10000 iterations as indicated on the X-axis. The Y-axis shows the accuracy values from 0 to 1. Obviously, BiLSTM achieves higher accuracy than other architectures. The justification behind this is that BiLSTM accesses the data sets from different directions, understands the context well and eliminates the ambiguity. Also, it is observed that LSTM achieves higher accuracy than GRU and LSTM + GRU architectures. In particular, the overall accuracy approximately increases by 20% per every 1000 iterations and BiLSTM achieves approximately double the accuracy compared with conventional LSTM.

#### 2) EXP2:- EFFECT OF CHANGING BACKPROPAGATION OPTIMIZERS ON ACCURACY & LOSS VALUES

In this set of experiments, the impact of selecting optimizers with RNN architectures is studied. This set of experiments compares two optimizers Adam optimizer and RMSProp Optimizers. In Figure 7 and Figure 8, the X-axis indicates the training iterations values from 1000 to 10000. The Y-axis shows the accuracy values from 0 to 1. Figure 7(a) shows that the RMSProp achieves higher accuracy with LSTM than Adam optimizer, while Figure 7(b) and Figure 7(c) show that Adam optimizer with GRU and LSTM + GRU achieves higher accuracy than RMSProp. Similarly, Figure 8(a) shows that the RMSProp achieves lower loss values with LSTM than Adam optimizer, whilst Figure 8(b) and Figure 8(c) show that Adam optimizer with GRU and LSTM + GRU achieves lower loss values than RMSProp. This ensures the strong relationship between the model accuracy and loss, the more accuracy achieved, the fewer loss values obtained. In a nutshell, the 20% accuracy increase is equivalent to 3 decreases in loss values.

#### 3) EXP3:- EFFECT OF USING SIMILARITY MODULE IN *DeepMotions*

In this set of experiments, Figure 9 compares the *DeepMotions* with and without designed Similarity Module. Figure 9 shows the *DeepMotions* with Similarity Module achieves higher accuracy than *DeepMotions* without Similarity Module. The justification behind this is that the *DeepMotions* with Similarity Module operates on a generated similar training set, as a result, the deep learning model is more generic and any ambiguity in the data is eliminated.

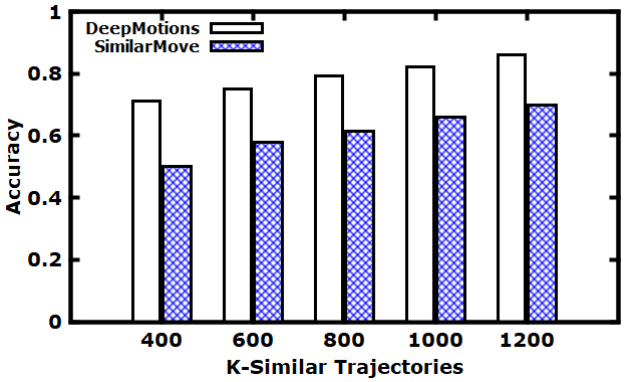



**FIGURE 10.** *DeepMotions* vs. *SimilarMove*.

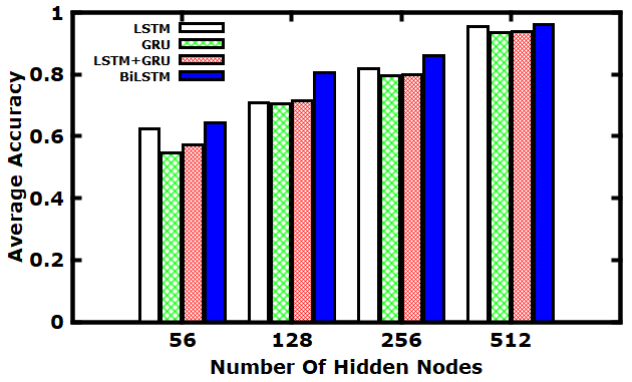



**FIGURE 12.** Hidden nodes (accuracy).

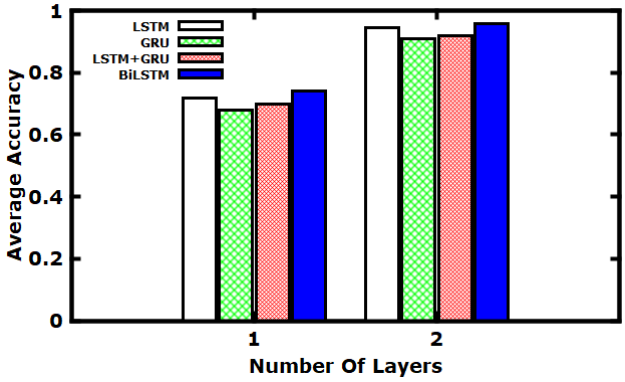



**FIGURE 11.** Hidden layers (accuracy).

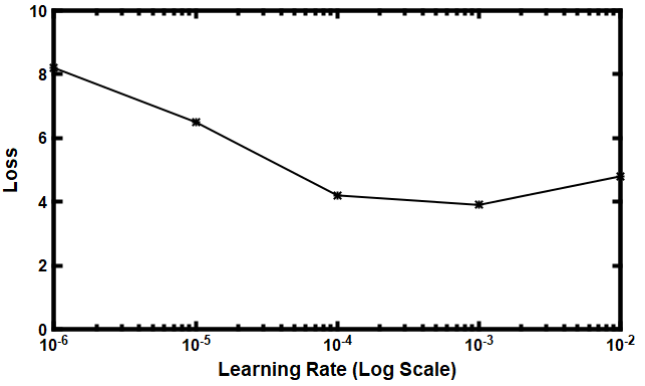



**FIGURE 13.** Learning rate (accuracy).

#### 4) EXP4:- EFFECT OF USING DEEP LEARNING IN *DeepMotions*

In this set of experiments, Figure 10 compares *SimilarMove* [2] System with *DeepMotions* system as both of them represents a similarity-based prediction system. The Prediction Building Module employed inside *SimilarMove* is implemented by using the Hidden Markov Model (HMM), while the Prediction Building Module employed inside *DeepMotions* is implemented using Bi-directional recurrent neural network. Accordingly, *SimilarMove* can be considered as a machine learning prediction system and *DeepMotions* can be considered as a deep learning prediction system. Overall, the accuracy increases as the number of similar trajectories increases. Figure 10 shows that *DeepMotions* system achieves an observed increase in accuracy more than *SimilarMove* system. More specifically, *DeepMotions* increases prediction accuracy around 15%-20% more than *SimilarMove*.

#### 5) EXP5:- EFFECT OF ADDING EXTRA HIDDEN LAYERS

In this set of experiments, Figure 11 investigates the impact of adding additional hidden layers with respect to accuracy. In general, increasing the number of hidden layers improves the accuracy, however, it is noticed that increasing the number of hidden layers much more than the 2 will cause the accuracy to decrease and make network overfit to the training set.

#### 6) EXP6:- EFFECT OF ADDING EXTRA HIDDEN NODES

In this set of experiments, Figure 12 presents the results obtained from the analysis of adding extra hidden nodes inside each hidden layer. From the graph, it is observed that overall increase the number of hidden nodes through the network make the accuracy gradually increased. More specifically, duplicating the number of hidden nodes improves the accuracy around 22%. It is important to note that no improvements happened in accuracy when increasing the number of hidden nodes much more than the 512.

#### 7) EXP7:- EFFECT OF CHANGING THE LEARNING RATE

In this set of experiments, Figure 13 examines the relationship between learning rate and loss. This set of experiments demonstrates how should to estimate a good learning rate by training the model initially with a very low learning rate (0.000001, 0.00001, 0.0001, 0.001, 0.01) and increasing it. Moreover, the learning rate implies how quickly our model can accomplish the best accuracy (a local minima). It is observed that when the learning rate increase, the loss value decreases. However, there will be a point 0.01 where the loss stops decreasing and starts to increase. From the practice, the learning rate should ideally be 0.001 to arrive at the local minima.

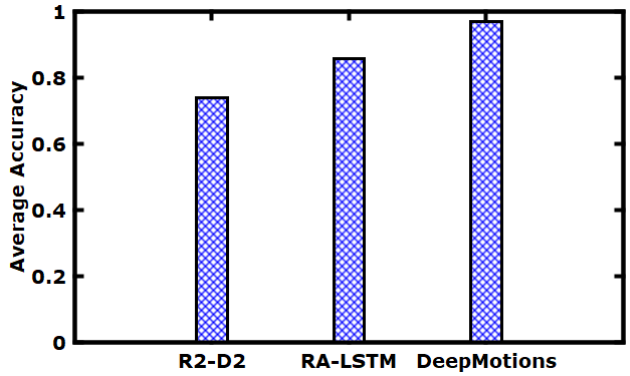



**FIGURE 14.** Comparison of baseline methods.

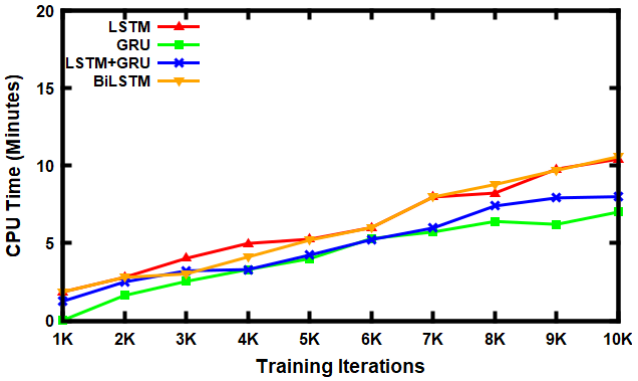



**FIGURE 15.** RNN architectures (CPU processing time).

#### 8) EXP8:- COMPARISON OF BASELINE METHODS

In this set of experiments, Figure 14 shows the comparison results between *DeepMotions* and 2 other models for predicting future trajectory path. It is observed that the average accuracy achieved by *DeepMotions* is greater than other competitive methods $R2-D2$ and $RA-LSTM$ by 15-20% approximately. The justification behind this is that due to several reasons such as; (1) *DeepMotions* removes irregularities in input data and model trained in similar data only, this make the model trains in an easy and accurate manner, and (2) the *DeepMotions* employs the Bi-LSTM during the prediction process which makes the model understands the context well rather than HMM and other RNN architectures.

### D. PERFORMANCE EVALUATION

This section evaluates the performance of our proposed system *DeepMotions* experimentally. Overall, the consumed CPU time processing increases when training iterations increase. Additionally, it is noticed that the Similarity Module plays a significant role in saving processing time as it filters out the non-similar data, and therefore the size of data set is less than the original data set.

#### 1) PERFORMANCE EVALUATION RESULTS

We next report our findings.

#### 2) EXP9:- EFFECT OF CHOOSING RNN ARCHITECTURE

In this set of experiments, Figure 15 compares the different RNN architectures;BiLSTM, LSTM, GRU, and

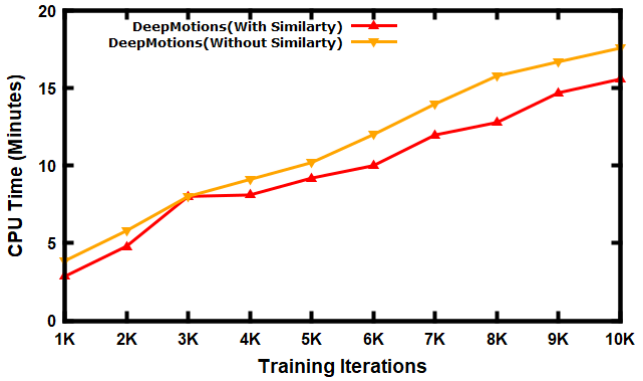



**FIGURE 16.** *DeepMotions* with similarity VS. *DeepMotions* without similarity (CPU processing time).

LSTM + GRU with respect to CPU processing time. Figure 15 shows the training iterations change from 1000 to 10000 iterations as indicated on the X-axis. The Y-axis shows the time in minutes from 0 to 20. It is observed that LSTM consumes more time than GRU, the justification behind this is the structure of GRU as it contains only two gates, while LSTM contains three gates. In BiLSTM the 10000 iterations consumed around 10.5 minutes, while LSTM consumes 10.4 minutes and GRU consumes 8 minutes for the same number of iterations. To conclude, in LSTM the average consumed time per 1000 iteration is 1.4 minute, the average consumed time per 1000 iteration in GRU is 0.8 minute and the average consumed time per 1000 iteration in BiLSTM is 1.5 minute.

#### 3) EXP10:- EFFECT OF USING SIMILARITY MODULE IN *DeepMotions*

In this set of experiments, Figure 16 compares the *DeepMotions* system with and without the Similarity Module. The x-axis indicates the training iterations from 1000 to 10000. The y-axis indicates the accuracy values from 0 to 1. It is noticed that the *DeepMotions* system with employed Similarity Module consumes less processing time than *DeepMotions* without using Similarity Module. The root cause of this is that Similarity Module in *DeepMotions* acts as a filter, so not all data sets are processed and similar data only that used during the prediction process is used which helps in saving CPU processing time. *DeepMotions* with employed Similarity Module saves approximately 2 minutes per 1000 iterations.

#### 4) EXP11:- EFFECT OF ADDING EXTRA HIDDEN LAYERS

In this set of experiments, Figure 17 shows the impact of adding additional hidden layers with respect to CPU processing time. In general, increasing the number of hidden layers consumes more time. In this case, choosing 2 layers to run in our model consumed around 25% CPU time greater than running the model with only one layer.

#### 5) EXP12:- EFFECT OF ADDING EXTRA HIDDEN NODES

In this set of experiments, Figure 18 shows the impact of adding extra hidden nodes with respect to CPU processing

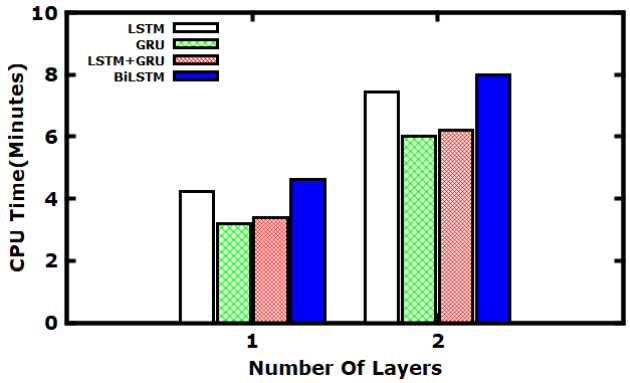



**FIGURE 17.** Hidden Layers (CPU processing time).

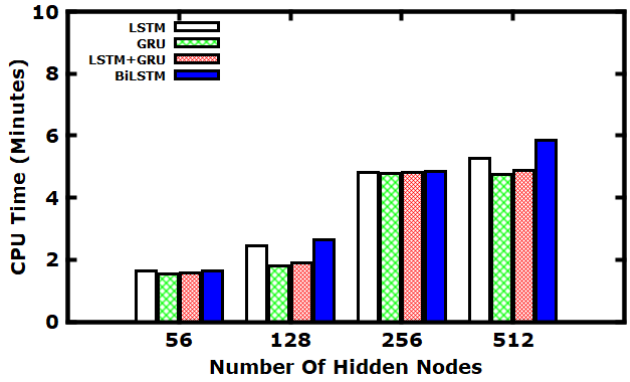



**FIGURE 18.** Hidden nodes (CPU processing time).

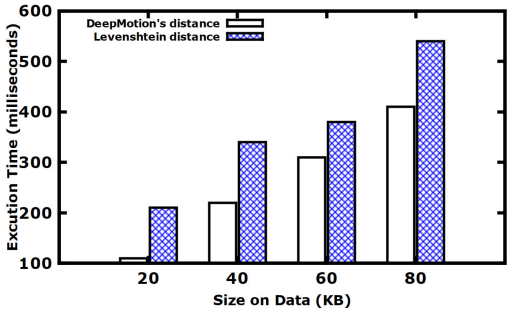



**FIGURE 19.** *DeepMotions's* edit distance VS. Levenshtein distance (excution time).

time. In general, increasing the number of hidden nodes consumes more time. In this case, the duplicating number of hidden nodes will cause increasing CPU time processing by 10-15% approximately.

#### 6) EXP13:- EFFECT OF USING THE DEVISED EDIT DISTANCE

In this set of experiments, Figure19 presents the impact of using the proposed edit distance measurement employed inside DeepMotions. Figure 22 compares the proposed edit distance by Levenshtein distance [30] with respect to the execution time of each other. It is observed that our proposed edit distance measurement consumes less execution time than Levenshtein distance by 40% approximately. The justification behind this is that our proposed measurement avoids more loops and comparisons.

### E. EXPERIMENTS SUMMARY

To sum up, the BiLSTM always achieves higher accuracy than LSTM and GRU. Also, LSTM achieves higher accuracy than GRU. The accuracy increases by increasing the training iterations. GRU always consumes less processing time than LSTM and Bi-directional. The RMSProp optimizer is the best choice for LSTM architecture, while Adam Optimizer is the best choice for GRU architecture. A strong relationship is discovered between deep learning model accuracy and loss when the accuracy increases the model loss must decrease.

## VII. CONCLUSION

In this paper, we addressed the problem of predicting the future paths of moving objects on a road network when their history is unavailable. For this purpose, we presented the *DeepMotions* system that predicts the future trajectory of a moving query object under the shortage of the object's movement history or past trajectories. The *DeepMotions* system takes an approach to retrieve the current moving objects' trajectories whose motion patterns are similar to that of a query object and employ these similar motions to infer the query object's future movements. The system is layered into three basic modules. The similarity module recognizes similar trajectories and excludes non-similar ones. Using the set of similar trajectories, the Prediction Building Module is constructed which will be queried by the Path Extractor Module to retrieve the possible future path along with its expected probability. Experimental results, based on real data, show that *DeepMotions* achieved high accurate prediction results with a minimal system overhead.

In the future work, we plan to build a generic and scalable framework that allows end-users to submit predictive queries on the cloud based through big data frameworks s. As a result, this framework must be able to manage, process and analyze the increasing volume of big spatial data. Additionally, we investigate to utilize GPU microprocessors in *DeepMotions* for better performance and efficient handling of multiple tasks simultaneously.

## REFERENCES

[1] A. M. Hendawi, M. Ali, and M. F. Mokbel, "Panda∗: A generic and scalable framework for predictive spatio-temporal queries," *GeoInformatica*, vol. 21, no. 2, pp. 175–208, 2012.
[2] M. Abdalla, A. Hendawi, N. ElGamal, H. M. O. Mokhtar, M. Ali, and J. Krumm, "SimilarMove: Similarity-based prediction for moving object future path," in *Proc. 2nd ACM SIGSPATIAL Workshop Predict. Hum. Mobility*, 2018, pp. 15–24.
[3] R. Amardeep and K. T. Swamy, "Training feed forward neural network with back propogation algorithm," *Int. J. Eng. Comput. Sci.*, vol. 6, no. 1, pp. 19860–19866, 2017.
[4] C. Song, Z. Qu, N. Blumm, and A.-L. Barabasi, "Limits of predictability in human mobility," *Science*, vol. 327, no. 5968, pp. 1018–1021, Feb. 2010.
[5] A. Prasad Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Modeling and querying moving objects," in *Proc. 13th Int. Conf. Data Eng.*, Nov. 2002, pp. 422–432.
[6] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: https://arxiv.org/abs/1412.3555
[7] B. Kim, C. M. Kang, J. Kim, S. H. Lee, C. C. Chung, and J. W. Choi, "Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network," in *Proc. IEEE 20th Int. Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2017, pp. 399–404.

[8] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 21–27, Jan. 1967.
[9] A. de Brébisson, É. Simon, A. Auvolat, P. Vincent, and Y. Bengio, "Artificial neural networks applied to taxi destination prediction," in *Proc. ECML-PKDD-DCs*, 2015, pp. 1–12.
[10] A. El-Shahat, "Introductory chapter: Artificial neural networks," in *Advanced Applications for Artificial Neural Networks*, 2018.
[11] M. Fathollahi and R. Kasturi, "Autonomous driving challenge: To infer the property of a dynamic object based on its motion pattern," in *Proc. Eur. Conf. Comput. Vis.* (Lecture Notes in Computer Science), 2016, pp. 40–46.
[12] I. Sasikala, M. Ganesan, and A. John, "Uncertain data prediction on dynamic road network," in *Proc. Int. Conf. Inf. Commun. Embedded Syst. (ICICES)*, Feb. 2014, pp. 1–4.
[13] A. M. Hendawi and M. F. Mokbel, "Panda: A predictive spatio-temporal query processor," In *Proc. 20th Int. Conf. Adv. Geographic Inf. Syst.*, Redondo Beach, CA, USA, 2012, pp. 13–22.
[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 15, no. 8, pp. 1735–1780, 1997.
[15] H. Jeung, Q. Liu, H. T. Shen, and X. Zhou, "A hybrid prediction model for moving objects," in *Proc. IEEE 24th Int. Conf. Data Eng.*, Apr. 2008, pp. 70–79.
[16] J. Zhou, A. K. H. Tung, W. Wu, W. S. Ng, "A 'semi-lazy' approach to probabilistic path prediction in dynamic environments," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 748–756.
[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.
[18] R. Benetis, C. S. Jensen, G. Karĉiauskas, and S. Ŝaltenis, "Nearest and reverse nearest neighbor queries for moving objects," *VLDB J.*, vol. 15, no. 3, pp. 229–249, Sep. 2006.
[19] H. A. Karimi and X. Liu, "A predictive location model for location-based services," in *Proc. of 11th ACM Int. Symp. Adv. Geographic Inf. Syst.*, 2003, pp. 126–133.
[20] J. Krumm, "Real time destination prediction based on efficient routes," SAE Tech. Paper, 2006.
[21] J. Krumm, "A Markov model for driver turn prediction," SAE Tech. Paper, 2008.
[22] J. Krumm, "Route prediction from trip observations," SAE Tech. Paper, 2008.
[23] L.-M. Fu, "Backpropagation in neural networks with fuzzy conjunction units," in *Proc. IJCNN Int. Joint Conf. Neural Netw.*, 1990, pp. 613–618.
[24] S.-W. Kim, J.-I. Won, J.-D. Kim, M. Shin, J. Lee, and H. Kim, "Path prediction of moving objects on road networks through analyzing past trajectories," in *Knowledge-Based Intelligent Information and Engineering Systems* (Lecture Notes in Computer Science), 2007, pp. 379–389.
[25] J. Krumm, J. Letchner, and E. Horvitz, "Map matching with travel time constraints," SAE Tech. Paper, 2007.
[26] S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, "Indexing the positions of continuously moving objects," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 331–342.
[27] J. Lv, Q. Li, Q. Sun, and X. Wang, "T-CONV: A convolutional neural network for multi-scale taxi trajectory prediction," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Jan. 2018, pp. 82–89.
[28] Q. Liu, S. Wu, L. Wang, and T. Tan, "Predicting the next location: A recurrent model with spatial and temporal contexts," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, p. 194–200.
[29] Y. Zheng, X. Xie, and W. Y. Ma, "GeoLife: A collaborative social networking service among user, location and trajectory," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, Jun. 2010.
[30] K. U. Schulz and S. Mihov, "Fast string correction with Levenshtein automata," *Int. J. Document Anal. Recognit.*, vol. 5, no. 1, pp. 67–85, Nov. 2002.
[31] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. 11th Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, 2010, pp. 1045–1048.
[32] A. M. Hendawi, J. Bao, M. F. Mokbel, and M. Ali, "Predictive tree: An efficient index for predictive queries on road networks," in *Proc. IEEE 31st Int. Conf. Data Eng.*, Apr. 2015, pp. 1215–1226.
[33] E. Necula, "dynamic traffic flow prediction based on GPS data," in *Proc. IEEE 26th Int. Conf. Tools Artif. Intell.*, Nov. 2014, pp. 922–929.
[34] *OpenStreetMap*. Accessed: Apr. 2018. [Online]. Available: https://www.openstreetmap.org/
[35] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.
[36] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu, "Prediction and indexing of moving objects with unknown motion patterns," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004, pp. 611–622.
[37] K. Raptopoulou, A. N. Papadopoulos, and Y. Manolopoulos, "Fast nearest-neighbor query processing in moving-object databases," *GeoInformatica*, vol. 7, no. 2, pp. 113–137, 2003.
[38] P. Hall, B. U. Park, and R. J. Samworth, "Choice of neighbor order in nearest-neighbor classification," *Ann. Statist.*, vol. 36, no. 5, pp. 2135–2152, Oct. 2008.
[39] S. Pattanayak, "Introduction to deep-learning concepts and tensorflow," in *Pro Deep Learning with TensorFlow*, 2017, pp. 89–152.
[40] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.
[41] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is 'nearest neighbor' meaningful?" in *Proc. Int. Conf. Database Theory*, 1999, pp. 217–235.
[42] N. T. Vu, P. Gupta, H. Adel, and H. Schutze, "Bi-directional recurrent neural network with ranking loss for spoken language understanding," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2016, pp. 6060–6064.
[43] B. Singh, T. K. Marks, M. Jones, O. Tuzel, and M. Shao, "A multi-stream bi-directional recurrent neural network for fine-grained action detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 1961–1970.
[44] A. Slowik and M. Bialko, "Training of artificial neural networks using differential evolution algorithm," in *Proc. Conf. Hum. Syst. Interact.*, May 2008, pp. 60–65.
[45] Statista. *Number of Smartphone Users Worldwide*. Accessed: May 2018. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
[46] Statista. *Smartwatch Unit Sales Worldwide From 2014 to 2018*. Accessed: May 2018. [Online]. Available: https://www.statista.com/statistics/538237/global-smartwatch-unit-sales/
[47] S. Tamura, "On interpretations of a feed-forward neural network," in *Proc. Int. Joint Conf. Neural Netw.*, May 1989, p. 584.
[48] J. Sun, D. Papadias, Y. Tao, and B. Liu, "Querying about the past, the present, and the future in spatio-temporal databases," in *Proc. 20th Int. Conf. Data Eng.*, Sep. 2004, pp. 202–213.
[49] Y. Tao, J. Sun, and D. Papadias, "Analysis of predictive spatio-temporal queries," *ACM Trans. Database Syst.*, vol. 28, no. 4, pp. 295–336, Dec. 2003.
[50] Y. Tao and D. Papadias, "Spatial queries in dynamic environments," *ACM Trans. Database Syst.*, vol. 28, no. 2, pp. 101–139, Jun. 2003.
[51] F. Wu, K. Fu, Y. Wang, Z. Xiao, and X. Fu, "A spatial-temporal-semantic neural network algorithm for location prediction on moving objects," *Algorithms*, vol. 10, no. 2, p. 37, Mar. 2017.
[52] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, 2016, pp. 207–212.
[53] J. Cui, X. Zhou, Y. Zhu, and Y. Shen, "A road-aware neural network for multi-step vehicle trajectory prediction," in *Database Systems for Advanced Applications* (Lecture Notes in Computer Science), 2018, pp. 701–716.
[54] S. Qiao, N. Han, W. Zhu, and L. A. Gutierrez, "TraPlan: An effective three-in-one trajectory-prediction model in transportation networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 3, pp. 1188–1198, Jun. 2015.

**MOHAMMED ABDALLA** received the master's degree in the field of data mining over spatial data. He is currently a Teacher Assistant with the Faculty of Computers and Artificial Intelligence, University of Beni-Souef. He taught multiple classes during his job, as a Teacher Assistant, like data mining, machine learning, information assurance, and systems artificial intelligence. His research interests include data science, big data systems, database management systems, and spatiotemporal databases.

**ABDELTAWAB HENDAWI** received the B.Sc. and M.Sc. degrees from Cairo University, and the M.Sc. and Ph.D. degrees in computer science and engineering from the University of Minnesota, Twin Cities. His research interests are centered on big data management and analytics with more focus on smart cities related applications. His work has been recognized by a number of awards, including the best paper/demo/poster awards from ACM SIGSPATIAL, the IEEE MDM, and Blue Sky Ideas.

**HODA M. O. MOKHTAR** received the B.Sc. (Hons.) and the M.Sc. degrees from the Department of Computer Engineering, Faculty of Engineering, Cairo University, in 1997 and 2000, respectively, and the Ph.D. degree in computer science from the Computer Science Department, University of California at Santa Barbara (UCSB), in 2005. She is the Chair of the Information Systems Department, Faculty of Computers and Artificial Intelligence, Cairo University. She taught multiple courses both for the undergraduate and graduate levels at the Faculty of Computers and Artificial Intelligence, Cairo University, where she supervised a number of master's and Ph.D. theses students. Her research interests include big data analytics, data warehousing, data mining, database systems, social network analysis, bioinformatics, and Web services. She received the Scholarship as well as the Dean's Fellowship from the Computer Science Department, UCSB, in 2000, for her Ph.D. degree. She participated in several national committees, and was awarded multiple awards and certificates for her academic achievements.

**NEVEEN ELGAMAL** received the master's and Ph.D. degrees in the field of data warehousing. She is currently an Assistant Professor with the Faculty of Computers and Artificial Intelligence, Cairo University. She taught multiple classes during her job, as an Assistant Professor, like database systems, distributed databases, and Internet applications. Her fields of interests are data warehousing, data integration, data science, and location-based services. She is currently building her knowledge in multiple fields like data science and location-based services.

**JOHN KRUMM** received the Ph.D. degree in robotics and a thesis on texture analysis in images from the School of Computer Science, Carnegie Mellon University, in 1993. He worked at the Robotics Center of the Sandia National Laboratories, Albuquerque, NM, USA, for the next four years, where his main projects were computer vision for object recognition for use in robots and vehicles. He has been at the Microsoft Research, Redmond, WA, USA, since 1997, where he is currently a Principal Researcher with the AI Lab. His research focuses on understanding peoples' location and how to use that information to benefit the user. In 2017, he received the ten-year impact award for a article on location privacy from the ACM UbiComp Conference. He holds 56 U.S. patents. He was the PC chair for UbiComp 2007, ACM SIGSPATIAL 2013, and ACM SIGSPATIAL 2014, served on the Editorial Board of the *IEEE Pervasive Computing Magazine* and as a Co-Editor-in-Chief for the *Journal of Location Based Services*. He currently serves on the Editorial Board of the *Journal of Location Based Services*, as an Associate Editor of the *ACM Transactions on Spatial Algorithms and Systems*, and on the Executive Committee of ACM SIGSPATIAL.

**MOHAMED ALI** received the Ph.D. degree in computer science from Purdue University. In 2006, he joined the SQL Server Group, Microsoft, and he started another journey in Microsoft Bing Maps, in 2011, where he tackled various types of spatial search queries. While at Microsoft, he has also been an affiliate of the University Washington, where he taught database, data streaming, and GIS classes. In 2014, he has joined the School of Engineering and Technology, UWT, as an Associate Professor of computer science. He also serves as the Director of the Graduate Studies, UWT. He is currently an Associate Professor with the School of Engineering and Technology, University of Washington Tacoma (UWT). His research interests include data science, big data systems, database management systems, and spatiotemporal databases.